Mixture of Experts Meets Decoupled Message Passing: Towards General and Adaptive Node Classification

Xuanze Chen*
Jiajun Zhou*†
Zhejiang University of Technology
Hangzhou, China
{chenxuanze,jjzhou}@zjut.edu.cn

Shanqing Yu
Zhejiang University of Technology
Hangzhou, China
yushanqing@zjut.edu.cn

Qi Xuan
Zhejiang University of Technology
Hangzhou, China
xuanqi@zjut.edu.cn

ABSTRACT

Graph neural networks excel at graph representation learning but struggle with heterophilous data and long-range dependencies. And graph transformers address these issues through self-attention, yet face scalability and noise challenges on large-scale graphs. To overcome these limitations, we propose *GNNMoE*[1], a universal model architecture for node classification. This architecture flexibly combines fine-grained message-passing operations with a mixture-of-experts mechanism to build feature encoding blocks. Furthermore, by incorporating soft and hard gating layers to assign the most suitable expert networks to each node, we enhance the model's expressive power and adaptability to different graph types. In addition, we introduce adaptive residual connections and an enhanced FFN module into *GNNMoE*, further improving the expressiveness of node representation. Extensive experimental results demonstrate that *GNNMoE* performs exceptionally well across various types of graph data, effectively alleviating the over-smoothing issue and global noise, enhancing model robustness and adaptability, while also ensuring computational efficiency on large-scale graphs.

CCS CONCEPTS

• **Computing methodologies → Neural networks**.

KEYWORDS

Graph Neural Network, Node Classification, Mixture of Experts

*Both authors contributed equally to this research.

†Corresponding Author.

[1]Code is available at https://github.com/GISec-Team/GNNMoE.

1 INTRODUCTION

Graph Neural Networks (GNNs) have emerged as a powerful tool for representation learning on graph-structured data, leveraging message-passing mechanisms to capture intricate structural information. Despite their success, traditional GNNs face notable challenges, particularly in heterophilous graphs where their underlying assumption of homophily often leads to suboptimal performance [11]. Moreover, GNNs are prone to the over-smoothing problem [9] when handling long-range dependencies, causing node representations to become indistinguishable after multiple layers of propagation, which reduces their expressiveness.

To address these limitations, Graph Transformer (GT) has gained traction as a promising alternative. By adopting self-attention mechanisms, GT efficiently captures long-range dependencies in graphs, alleviating the over-smoothing issues inherent in traditional GNNs. Moreover, it demonstrates robustness in dealing with both structural and attribute heterogeneity, showing notable performance improvements in various tasks. However, despite its excellent performance in certain tasks, GT still faces several critical issues, primarily in two aspects. First, the self-attention mechanism, while powerful, may inadvertently introduce significant irrelevant global noise. This issue becomes pronounced in complex graph structures or graphs with weak node relationships, undermining the model's overall effectiveness. Second, the computational complexity of self-attention scales quadratically with graph size, posing severe bottlenecks for processing large-scale graphs and limiting GT's scalability in real-world applications. For the former issue, several studies [2, 4] incorporate topological information as masks or biases to recalibrate self-attention distributions, effectively reducing noise to some extent. For the latter, several studies [8, 10] either optimize the complexity of self-attention computation or compress the graph size to improve scalability. However, despite the observed performance improvement, these studies persist in applying the GT architecture to node classification tasks without thoroughly investigating whether the current GT architecture is genuinely adaptability for this purpose.

Currently, researchers often rely on prior knowledge of graph type to design specific graph representation learning models, requiring extensive experimentation to fine-tune architectures and parameters. This practice, while effective, lacks flexibility and makes it challenging to generalize across diverse graph types efficiently. Therefore, the development of a universal, adaptive, and computationally efficient graph model architecture is a pressing need. Such a model should excel in both homophilous and heterophilous settings, address over-smoothing, minimize global noise from self-attention, and maintain scalability for large-scale graphs.
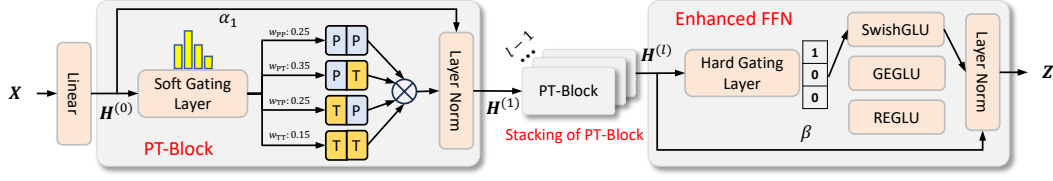
Figure 1: Illustration of *GNNMoE* architectures.

Building upon this motivation, we propose ***GNNMoE***, a universal node classification model designed to overcome these challenges. Our architecture decouples the message-passing process, allowing for flexible combinations of fine-grained operations tailored to different graph types. Inspired by the mixture of experts (MoE) [3], *GNNMoE* introduces an adaptive architecture search mechanism that assigns the most suitable expert combinations to each node, thereby enhancing expressiveness and adaptability. Additionally, drawing from residual connections and feed-forward network (FFN) designs in Transformer architectures, we incorporate adaptive residual connections and enhanced FFN modules into *GNNMoE*. These additions further refine the node representation process, boosting classification performance. Extensive experiments demonstrate that our *GNNMoE* consistently outperforms existing methods across a diverse range of graph datasets. Whether dealing with homophilous or heterophilous graphs, our framework effectively alleviates global noise and over-smoothing issues, showcasing superior robustness and adaptability. Furthermore, through its flexible message-passing operations and adaptive expert combination, *GNNMoE* achieves high performance in node classification tasks while maintaining computational efficiency, even for large-scale graphs.

2 PRELIMINARIES

A graph is denoted as $G = (V, E, X, Y)$, where V and E are the set of nodes and edges respectively, $X \in \mathbb{R}^{|V| \times d}$ is the node feature matrix, and $Y \in \mathbb{R}^{|V| \times C}$ is the node label matrix. Here we use $|V|$, d and C to denote the number of nodes, the dimension of the node features, and the number of classes, respectively. The graph topology information (V, E) can also be denoted by an adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$, where $A_{ij} = 1$ indicates the existence of an edge between v_i and v_j, and $A_{ij} = 0$ otherwise. Node classification is a fundamental task in graph machine learning, and it involves assigning labels to the nodes of a graph based on their features and the graph topology structure.

From a decoupled perspective, message passing in GNNs can be decomposed into two functionally independent operations [9]:

$$\text{propagation: } h_i^{(l)} = \text{P}\left(h_i^{(l-1)}, \left\{h_j^{(l-1)} \mid j \in \mathcal{N}(i)\right\}\right)$$
$$\text{transformation: } h_i^{(l)} = \text{T}\left(h_i^{(l)}\right) \tag{1}$$

where $h_i^{(l)}$ is the node representation during l-th message passing, $\mathcal{N}(i)$ is the neighbor set of node v_i. P is the propagation function that combines message generation and aggregation from neighbor node v_j to target node v_i. T performs a non-linear transformation on the state of the nodes after propagation. Based on the disentanglement, existing GNN architectures can be loosely categorized

into four types according to the stacking order of propagation and transformation operations: **PTPT**, **PPTT**, **TTPP**, and **TPTP**.

3 METHOD: GNNMOE

To achieve universal node classification across different graphs, we propose *GNNMoE*, as illustrated in Figure 1. This framework combines the advantages of both GTs and GNNs, consisting of stackable PT-Blocks and enhanced FFN. It takes node features and adjacency information as input and outputs the final node representations. The main highlights are reflected in the following aspects: 1) utilizing a soft gating mechanism to adaptively combine contributions from different message passing experts for each node, enabling flexible processing of different graph messages; 2) employing a hard gating mechanism to select appropriate activation layers, enhancing the expressiveness of FFN; 3) utilizing adaptive residual connections to improve adaptability to different data characteristics.

First, the input features X will be transformed into an initial feature embedding through a linear transformation parameterized by $W_0 \in \mathbb{R}^{d \times d'}$ and a ReLU activation:

$$H^{(0)} = \text{ReLU}\left(X W_0\right) \tag{2}$$

where d' is the hidden dimension. Next, we stack a series of message passing blocks, called PT-blocks, to further learn node representations. Each PT-block consists of a soft gating layer SG$(\cdot)$, an expert network $\mathcal{E}(\cdot)$, a layer normalization operation LN$(\cdot)$, and an adaptive initial residual connection, where the expert network $\mathcal{E} = \{PP, PT, TP, TT\}$ contains four message passing experts specialized in handling different graph features. For the (l)-th PT-block, it takes the node representation output from the $(l-1)$-th PT-block as input, then calculates the allocation weights of the expert network through the soft gating mechanism:

$$w_{\text{sg}} = \text{SG}\left(H^{(l-1)}\right) = \text{Softmax}\left(W_2 \cdot \text{ReLU}\left(H^{(l-1)} W_1\right)\right) \tag{3}$$

where $w_{\text{sg}} \in \mathbb{R}^4$ is the allocation weights, W_1 and W_2 are the transformation weights. Next, the graph messages processed by different experts are aggregated using allocation weights, and new node representations are generated through residual connections:

$$H^{(l-1)} = \sum_{i=1}^{4} w_{\text{sg}}^i \cdot \mathcal{E}_i\left(A, H^{(l-1)}\right)$$
$$H^{(l)} = \text{LN}\left(\alpha_l \cdot H^{(0)} + (1 - \alpha_l) \cdot H^{(l-1)}\right) \tag{4}$$

where α_l is a learnable parameter that controls the adaptive initial residual connection.

After message passing via l PT-blocks, *GNNMoE* has effectively fused the attribute information of the nodes with the topological information. Furthermore, inspired by the vanilla GT architecture,

Table 1: Node classification results: average test accuracy (%) ± standard deviation. The best results are highlighted in bold, while the second-best results are underlined. "Local Rank" indicates the average performance ranking across homophilous or heterophilous datasets, "Global Rank" indicates the average performance ranking across all datasets.

Method (Dataset)		Computers	Photo	Coauthor CS	Coauthor Physics	Facebook	ogbn-arxiv	Local Rank	Actor	Chameleon-fix	Squirrel-fix	Tolokers	Roman-empire	Penn94	Local Rank	Global Rank
Vanilla	MLP	85.01 ± 0.84	92.00 ± 0.56	94.80 ± 0.35	96.11 ± 0.14	76.86 ± 0.34	53.46 ± 0.35	18.67	37.14 ± 1.06	33.31 ± 2.32	34.47 ± 3.09	53.18 ± 6.35	65.98 ± 0.43	75.18 ± 0.35	17.33	18.00
	GCN	91.17 ± 0.54	94.26 ± 0.59	93.40 ± 0.45	96.37 ± 0.20	93.98 ± 0.34	69.71 ± 0.18	15.33	30.65 ± 1.06	41.85 ± 3.22	33.89 ± 2.61	70.34 ± 1.64	50.76 ± 0.46	80.45 ± 0.27	18.00	16.67
	GAT	91.44 ± 0.43	94.42 ± 0.61	93.20 ± 0.64	96.28 ± 0.31	94.03 ± 0.36	70.03 ± 0.42	14.50	30.58 ± 1.18	43.31 ± 3.42	36.27 ± 2.12	79.93 ± 0.77	57.34 ± 1.81	78.10 ± 1.28	16.67	15.58
	GraphSAGE	90.94 ± 0.56	95.41 ± 0.45	94.17 ± 0.46	96.69 ± 0.23	93.72 ± 0.35	69.15 ± 0.18	14.17	37.60 ± 0.95	44.94 ± 3.67	36.61 ± 3.06	82.37 ± 0.64	77.77 ± 0.49	OOM	10.83	12.50
Hetero-GNN	H2GCN	91.69 ± 0.33	95.59 ± 0.48	95.62 ± 0.27	97.00 ± 0.16	94.36 ± 0.32	OOM	7.00	37.27 ± 1.27	43.09 ± 3.85	40.07 ± 2.73	81.34 ± 1.16	79.47 ± 0.43	75.91 ± 0.44	10.83	8.92
	GPRGNN	91.80 ± 0.55	95.44 ± 0.33	95.17 ± 0.34	96.94 ± 0.20	94.84 ± 0.24	69.95 ± 0.19	6.67	36.89 ± 0.83	44.27 ± 5.23	40.58 ± 2.00	73.84 ± 1.40	67.72 ± 0.63	84.34 ± 0.29	9.50	8.08
	FAGCN	89.54 ± 0.75	94.44 ± 0.62	94.93 ± 0.22	96.91 ± 0.27	91.90 ± 1.95	66.87 ± 1.48	15.33	37.59 ± 0.95	45.28 ± 4.33	41.05 ± 2.67	81.38 ± 1.34	75.83 ± 0.35	79.01 ± 1.09	8.17	11.75
	ACMGCN	91.66 ± 0.78	95.42 ± 0.39	95.47 ± 0.33	97.00 ± 0.27	94.27 ± 0.33	69.98 ± 0.11	7.83	36.89 ± 1.13	43.99 ± 2.02	36.58 ± 2.75	83.52 ± 0.87	81.57 ± 0.35	83.01 ± 0.46	11.00	9.42
	FSGNN	91.03 ± 0.56	95.50 ± 0.41	95.51 ± 0.32	96.98 ± 0.20	94.32 ± 0.32	71.09 ± 0.21	7.33	37.14 ± 1.06	45.79 ± 3.31	38.25 ± 2.62	83.87 ± 0.98	79.76 ± 0.41	83.87 ± 0.98	8.00	7.67
	LINKX	90.75 ± 0.36	94.58 ± 0.56	95.52 ± 0.30	96.93 ± 0.16	93.84 ± 0.32	66.16 ± 0.27	12.17	31.17 ± 0.61	44.94 ± 3.08	38.40 ± 3.54	77.55 ± 0.80	61.36 ± 0.60	84.97 ± 0.46	12.50	12.33
GT	Vanilla GT	84.41 ± 0.72	91.58 ± 0.73	94.61 ± 0.30	OOM	OOM	OOM	19.67	37.08 ± 1.08	44.27 ± 3.98	39.55 ± 3.10	72.24 ± 1.17	OOM	OOM	14.33	17.00
	ANS-GT	90.01 ± 0.38	94.51 ± 0.24	93.93 ± 0.23	96.28 ± 0.19	92.61 ± 0.16	OOM	17.50	37.80 ± 0.95	40.74 ± 2.26	36.65 ± 0.80	76.91 ± 0.85	80.36 ± 0.71	OOM	13.00	15.25
	NAGFormer	90.22 ± 0.42	94.95 ± 0.52	94.96 ± 0.25	96.43 ± 0.20	93.35 ± 0.28	70.25 ± 0.13	13.83	36.99 ± 1.39	46.12 ± 2.25	38.31 ± 2.43	66.73 ± 1.18	75.92 ± 0.69	73.98 ± 0.53	13.00	13.42
	SGFormer	90.70 ± 0.59	94.46 ± 0.49	95.21 ± 0.20	96.87 ± 0.18	86.66 ± 0.54	66.83 ± 0.21	15.00	36.59 ± 0.90	44.27 ± 3.68	38.83 ± 2.19	80.46 ± 0.91	76.41 ± 0.50	76.65 ± 0.49	13.17	14.08
	Exphormer	91.46 ± 0.51	95.42 ± 0.26	95.62 ± 0.29	96.89 ± 0.20	93.88 ± 0.40	71.59 ± 0.24	8.17	36.83 ± 1.10	42.58 ± 3.24	36.19 ± 3.20	82.26 ± 0.41	87.55 ± 1.13	OOM	13.33	10.75
	Difformer	91.52 ± 0.55	95.41 ± 0.38	95.49 ± 0.26	96.98 ± 0.22	94.23 ± 0.47	OOM	9.83	36.73 ± 1.27	44.44 ± 3.20	40.45 ± 2.51	81.04 ± 4.16	72.52 ± 0.44	OOM	12.67	11.25
GNNMoE	GCN-like P	92.17 ± 0.50	95.81 ± 0.41	95.81 ± 0.26	97.03 ± 0.13	95.53 ± 0.35	72.29 ± 0.16	1.33	37.59 ± 1.36	47.19 ± 2.93	44.02 ± 2.59	84.77 ± 0.93	85.05 ± 0.55	85.11 ± 0.39	2.33	1.83
	SAGE-like P	91.85 ± 0.39	95.46 ± 0.24	95.68 ± 0.24	96.81 ± 0.22	94.63 ± 0.36	71.94 ± 0.25	5.67	37.97 ± 1.01	45.73 ± 3.19	39.19 ± 2.84	83.96 ± 0.75	86.00 ± 0.45	84.05 ± 0.37	4.17	4.92
	GAT-like P	91.98 ± 0.46	95.71 ± 0.37	95.72 ± 0.23	97.05 ± 0.19	95.21 ± 0.25	72.45 ± 0.32	1.67	37.76 ± 0.98	45.56 ± 3.94	39.19 ± 2.38	85.45 ± 0.94	87.29 ± 0.60	81.98 ± 0.47	4.33	3.00
GNNMoE (GCN-like P)	w/o FFN	91.29 ± 0.36	95.67 ± 0.42	95.49 ± 0.25	96.95 ± 0.17	94.98 ± 0.37	71.31 ± 0.29	6.17	37.26 ± 0.91	45.22 ± 4.21	39.08 ± 2.19	84.17 ± 0.70	84.53 ± 0.26	79.80 ± 1.21	7.67	6.92
	w/o AIR/AR	91.76 ± 0.32	95.33 ± 0.49	94.31 ± 0.31	96.75 ± 0.27	94.91 ± 0.40	70.88 ± 0.28	9.83	37.35 ± 0.98	47.09 ± 3.39	39.15 ± 2.87	84.04 ± 0.70	84.88 ± 0.84	80.61 ± 0.97	5.83	7.83

where adding FFN can enhance the expressiveness of vanilla GNN, we design an enhanced FFN module in the *GNNMoE* architecture. Specifically, the enhanced FFN module consists of a hard gating layer HG(·), an expert network $\mathcal{A}(\cdot)$, a layer normalization operation, and an adaptive residual connections, where the expert network $\mathcal{A} = \{\text{SwishGLU}, \text{GEGLU}, \text{REGLU}\}$ contains three activation function experts. SwishGLU combines Swish activation with gating mechanisms to promote more effective gradient propagation; GEGLU enhances nonlinear expressiveness through additive activation and gating; REGLU introduces gating on top of ReLU to reduce gradient vanishing and improve computational efficiency.

In the enhanced FFN, the node features encoded by l PT-blocks are first input into a hard gating layer, which selects the appropriate activation function expert for further feature encoding:

$$j = \text{HG}\left(H^{(l)}\right) = \text{Gumbel\_Softmax}\left(H^{(l)}\right) \in \{1, 2, 3\} \quad (5)$$

Then, the selected expert will further encode $H^{(l)}$ to enhance its expressiveness, followed by an adaptive residual connection to generate the final node representation:

$$Z = \mathcal{A}_j\left(H^{(l)}\right) = \left(\sigma_j\left(H^{(l)}W_3\right) \otimes H^{(l)}W_4\right)W_5$$
$$Z = \text{LN}\left(\beta \cdot H^{(0)} + (1 - \beta) \cdot Z\right) \quad (6)$$

where $\sigma \in \{\text{Swish}, \text{GELU}, \text{ReLU}\}$, $W_3, W_4, W_5 \in \mathbb{R}^{d' \times d'}$ are the transformation weights, $\otimes$ is the element-wise multiplication, β is a learnable parameter that controls the adaptive residual connection.

To achieve node classification, we finally use a prediction head f_{pred} parameterized by $W_6 \in \mathbb{R}^{d' \times C}$ and Softmax activation to obtain the node predictions. During model training, binary cross-entropy classification loss is used as the optimization objective.

$$\hat{Y} = \text{Softmax}\left(ZW_6\right), \quad \mathcal{L} = -\text{trace}\left(Y_{\text{train}}^{\top} \cdot \log \hat{Y}_{\text{train}}\right) \quad (7)$$

where the trace operation trace (·) is used to compute the sum of the diagonal elements of the matrix.

4 EXPERIMENTS

4.1 Experiment Settings

4.1.1 Datasets and Baselines. We conduct extensive experiments on 12 benchmark datasets, which include (1) Six homophilous datasets: Computers, Photo, Coauthor CS, Coauthor Physics, Facebook and ogbn-arxiv; and (2) Six heterophilous datasets: Actor, Squirrel-fix, Chameleon-fix [5], Tolokers, Roman-empire and Penn94. All datasets are divided into training, validation, and testing sets in a proportion of 48%: 32%: 20%. We compare *GNNMoE* with three kinds of baselines, which include (1) Vanilla model: MLP, GCN, GAT, GraphSAGE; (2) Heterophilous GNNs: LINKX, H2GCN, GPRGNN, FAGCN, ACMGCN, FSGNN; and (3) GT models: vanilla GT, ANS-GT, NAGFormer [1], SGFormer [8], Exphormer [6] and Difformer [7].

4.1.2 Experimental Settings. We utilize 10 random seeds to fix the data splits and model initialization, and report the average accuracy and standard deviation over 10 runs. For all methods, we set the search space of common parameters as follows: maximum epochs to 500 with 100 patience, hidden dimension d' to 64, optimizer to AdamW, learning rate in {0.005, 0.01, 0.05, 0.1}, dropout rate in {0.1, 0.3, 0.5, 0.7, 0.9}. For *GNNMoE*, the number of PT-blocks in {3,4,5,6} is searched for ogbn-arxiv while a fixed value of 2 is used for other datasets. For all baselines, we search the common parameters in the same parameter spaces.

4.2 Evaluation on Node Classification

Table 1 reports the node classification results of all methods, from which we can draw the following conclusions: 1) *GNNMoE* consistently demonstrates higher local and global average performance rankings on both homophilous and heterophilous datasets, indicating its effectiveness, superiority, and stability in node classification tasks, significantly surpassing three categories of baselines; 2) Our method successfully avoids encountering out-of-memory (OOM) issues, in contrast to certain GT-based methods and spatial-domain
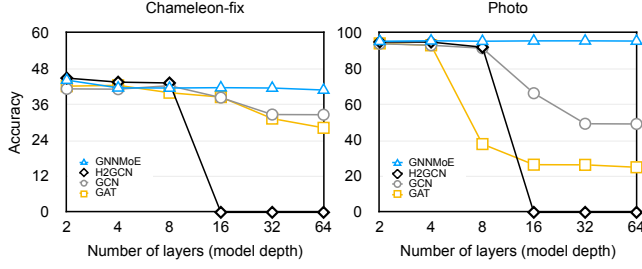
Figure 2: Impact of model depth.

Figure 3: Efficiency analysis on ogbn-arxiv dataset.

GNNs, which suggests the architectural efficiency of *GNNMoE* and its scalability in large-scale graph computations.

We further conduct ablation studies to analyze the impact of specific components on the performance of *GNNMoE*. As observed, removing the FFN module (w/o FFN) and the residual connection module (w/o AIR/AR) leads to significant performance degradation across all datasets. This highlights the critical role of the enhanced FFN module in improving *GNNMoE*'s utilization of important features, as well as the contribution of the residual connection module in facilitating the model's use of initial information. Both components contribute to *GNNMoE*'s universality across different graphs.

4.3 More Analysis

4.3.1 Impact on Model Depth. Figure 2 demonstrates the impact of model depth on performance. It is evident that Vanilla GNN's performance rapidly deteriorates as model depth increases, indicating the presence of over-smoothing. Meanwhile, H2GNN's performance gradually declines as model depth increases from 2 to 8 layers, and encounters memory overflow when model depth exceeds 16 layers. In contrast, our method maintains consistently stable performance while stacking PTblock message passing modules, demonstrating its immunity to the over-smoothing problem.

4.3.2 Efficiency Analysis. Figure 3 illustrates the efficiency and accuracy of several representative methods on the ogbn-arxiv dataset, where the x-axis represents the number of epochs at which early stopping is triggered, the y-axis represents the total training time, and the bubble size reflects accuracy. As shown, compared to the spatial-domain GNN method FSGNN and some GT-based methods, *GNNMoE* consumes 2-7 times less training time. Additionally, compared to traditional GNN methods, *GNNMoE* converges in fewer epochs. In summary, *GNNMoE* demonstrates good computational efficiency while maintaining high performance.

5 CONCLUSION

We combine the strengths of GNNs and GTs to design a universal node classification model architecture, *GNNMoE*. This architecture encapsulates adaptive message passing into expert network blocks, providing flexible encoding capabilities for different types of graphs. Extensive experiments demonstrate that, compared to existing GNNs, as well as GTs, *GNNMoE* outperforms in node classification performance, while also adapting to various graph types, showcasing its universality. Additionally, our architecture effectively addresses inherent challenges such as over-smoothing and
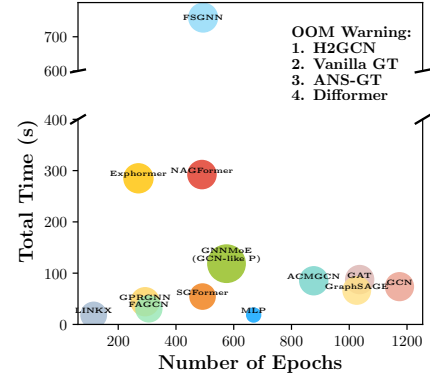
inefficiency. In the future, we will further improve the existing architecture by optimizing both the input and structure of the gating networks, while also exploring graphs in more domains.

ACKNOWLEDGMENTS

REFERENCES

[1] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. 2023. NAGphormer: A Tokenized Graph Transformer for Node Classification in Large Graphs. In *Proceedings of the International Conference on Learning Representations*. 1–18.

[2] Vijay Prakash Dwivedi and Xavier Bresson. 2021. A Generalization of Transformer Networks to Graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications* (2021), 1–8.

[3] Fenyu Hu, Liping Wang, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2022. GraphDIVE: Graph Classification by Mixture of Diverse Experts. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*. 2080–2086.

[4] Erxue Min, Runfa Chen, Yatao Bian, Tingyang Xu, Kangfei Zhao, Wenbing Huang, Peilin Zhao, Junzhou Huang, Sophia Ananiadou, and Yu Rong. 2022. Transformer for graphs: An overview from architecture perspective. *arXiv preprint arXiv:2202.08455* (2022).

[5] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. 2023. A Critical Look at The Evaluation of GNNs under Heterophily: Are We Really Making Progress?. In *The Eleventh International Conference on Learning Representations*. 1–15.

[6] Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. 2023. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*. PMLR, 31613–31632.

[7] Qitian Wu, Chenxiao Yang, Wentao Zhao, Yixuan He, David Wipf, and Junchi Yan. 2023. DIFFormer: Scalable (Graph) Transformers Induced by Energy Constrained Diffusion. In *International Conference on Learning Representations (ICLR)*. 1–26.

[8] Qitian Wu, Wentao Zhao, Chenxiao Yang, Hengrui Zhang, Fan Nie, Haitian Jiang, Yatao Bian, and Junchi Yan. 2024. Simplifying and empowering transformers for large-graph representations. *Advances in Neural Information Processing Systems* 36 (2024), 64753–64773.

[9] Wentao Zhang, Zeang Sheng, Ziqi Yin, Yuezihan Jiang, Yikuan Xia, Jun Gao, Zhi Yang, and Bin Cui. 2022. Model degradation hinders deep graph neural networks. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*. 2493–2503.

[10] Zaixi Zhang, Qi Liu, Qingyong Hu, and Chee-Kong Lee. 2022. Hierarchical graph transformer with adaptive node sampling. *Advances in Neural Information Processing Systems* 35 (2022), 21171–21183.

[11] Jiajun Zhou, Chenxuan Xie, Shengbo Gong, Jiaxu Qian, Shanqing Yu, Qi Xuan, and Xiaoniu Yang. 2024. PathMLP: Smooth path towards high-order homophily. *Neural Networks* 180 (2024), 106650.